***BSc.(Information Technology) (Semester III) 2018-19***

***Data Structures***

***(USIT 302 Core)***

***University Paper Solution***

***By***

***Mr. Shajil Kumar P A***

Mr. Shajil Kumar P A Page 1

***Question 1***

***Q1a. What is an algorithm? Explain properties of an algorithm?***

***Ans:*** An algorithm can be defined as finite collection of well defined steps designed to solve a particular problem. An algorithm must have following characteristics

**Input** – Must take some inputs that are required for the solution of a problem

**Process** – Must perform certain operations on the input data which is necessary for the solution of the problem

**Output** – Should produce certain output after processing the inputs

**Finiteness** – Must terminate after executing certain finite number of steps

**Effectiveness** – Every step should play a role in the solution to problem where each step must be unambiguous, feasible and definite

***Q1b. Write an algorithm for searching the element in an array***

***Ans:*** Searching operation refers to finding the position of desired element in an array. We can find the position of a given data element using two different searching techniques

**Linear Search**: - In the linear search technique we start comparing the desired element with the element at the first position of the array, then with the element at the second position in the array. This procedure continues until the desired element is found or we reach at the end of the array. If we don’t find the desired element till the end of the array it is concluded that the array is not having the desired element. If we find the desired element then we display the position in the array where the element is located

**Algorithm**

Step 1 Repeat Steps 2 and 3 For i =1 to n

Step 2If S[i] = Data then

Print “Element is found at position” : i

Exit

[End If]

Step 3. Set i=i+1 [End Loop]

Step 4. Print: “Desired element Data is not found in the array”

Step 5. Exit

**Binary Search**: - It is applied on the array where the elements are sorted alphabetically or numerically. The first step is to find the array index of the middle element. Then we compare the array element present at the middle with the desired element. If the element at the middle index position is the desired element then stop searching. Otherwise after examining the element at the middle index position we decide which half portion of the array may contain the desired element.

Mr. Shajil Kumar P A Page 2

**Algorithm**

Step 1. Set Start = lb , End =ub

Step 2. Repeat Steps 3 to 5 while Start < End

Step 3. Set Middle = Integer ((start+end)/2)

Step 4. If S[middle] =Data then

Print: “Element is found at position”: Middle

Exit

[End If]

Step 5. If S[Middle]< Data Then

Set Start =Middle + 1

Else

Set End = Middle -1

[End If]

[End Loop]

Step 6. Print: Element does not exit in the array”

Step 7. Exit

***Q 1c. What is data structure? Explain primitive and non-primitive data structure.***

***Ans:*** A data structure is a way of storing the data in computer’s memory so that it can be used efficiently. A data structure is a logical/ mathematical model of organization of data.

The choice of data structure begins with the choice of an Abstract Data Type(ADT) such as array.

**Primitive Data types** – Primitive data structures are also known as predefined or basic data structures. These data structures are different for different languages. For example in C language, the primitive data types for storing the integer values are int, long, short and for storing the real values are float, double and long double.

**Abstract Data Types (ADT)** – Abstract stands for considering apart from the details specification or implementation. Abstraction refers to the act of representing the essential features without including the details (hiding the details same as OOPS). Encapsulation refers to providing data and operations on the data in a single unit (same as OOPS). The examples of abstract data types are stack, queue, tree etc.

***Q1d. What is time and space complexity? Explain Big O and Big Theta notation***

***Ans:*** Complexity is the time (Amount of time to execute) and space (Space required to store variables and constants and instruction sets) requirement of the algorithm. If the time and space requirement is more than the complexity of the algorithm is more and vice versa. The space requirement of the algorithm is not a very important factor because it is available at very low cost. The time requirement of the algorithm is considered an important factor and hence termed as time complexity.

**Big O notation**: - Big O notation is **upper bound** asymptotic notation. This means, a function f(x) is a Big O of function g(x) and there exists the positive constants **c** and **n0** such that

**c.g(n) >= f(n) for all n>=n0**where f(x) and g(x) are the functions of non-negative integers.

**Big Theta notation**: - Big Theta notation is **tight bound** asymptotic notation. This means, a function f(x) is a Big Theta of function g(x) and there exists three positive constants **c1, c2** and **n0** such that

**c1.g(n) <= f(n) <= c2.g(n) for all n>=n0**where f(x) and g(x) are the functions of non-negative integers.

Mr. Shajil Kumar P A Page 3

***Q1e. Write an algorithm for sorting the elements of an array.***

***Ans:*** Sorting an array refers to arranging the elements of an array in some logical order, which can be increasing or decreasing of elements. The sorting technique which is popularly used is **Exchange Sort or Bubble Sort**. The algorithm starts with comparing the first element with the second element and exchange takes place only if the first element is larger than the second element. Then the second element is compared with the third element and so on. This process goes on until the array is sorted

**Algorithm**

Step 1.Repeat For p=1 to n-1

Step 2. For i=1 to n-p

Step 3. If S[i]>S[i+1] Then

Exchange S[i] with S[i+1]

[End If]

[End Loop]

[End Loop]

Step 4. Exit

***Q1f. Write an algorithm for merging two arrays***

***Ans:*** Merging of arrays refers to combining the elements of two linear arrays into a single array.

**Algorithm**

Step 1.Set i=lb1, j=lb2, k=1

Step 2.While i<= ub1 AND j<= ub2

Step 3. If A1[i] < A2[j] Then

Set A3[k] =A1[i]

Set i=i+1

Set k=k+1

Else

Set A3[k] =A2[j]

Set j=j+1

Set k=k+1

[End If]

[End Loop]

Step 4. If i> ub1 Then

While j<= ub2

Set A3[k]=A2[j]

Set j=j+1

Set k=k+1

[End Loop]

Else If j>ub2 Then

While i<=ub1

Set A3[k] =A1[i]

Set i=i+1

Set k=k+1

[End Loop]

[End If]

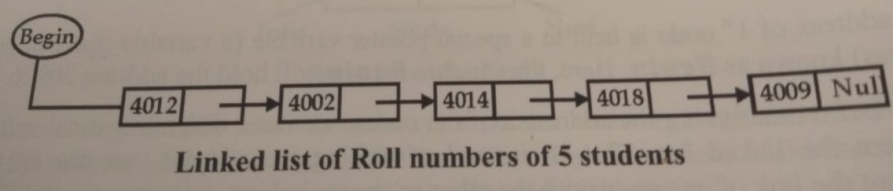
Step 5. Exit

Mr. Shajil Kumar P A Page 4

***Question 2***

***Q2a. Explain the structure of single linked list.***

***Ans:*** A linked list can be defined as the linear collection of elements where each element is stored in a node and the linear order between elements is given by the means of pointers instead of sequential memory locations. In linked list or one way list, each node is divided into two parts. The first part of the node contain the element itself and the second part which is termed as next field or pointer field contain the address of the next node in the list. One-way linked list is also known as singular linked list. In this each node has at least two parts. The first part is known as **Info** part which hold the element and the second part is known as **Next** part which hold the address of next node. The address of first node is stored in as special variable known as **Begin** and the **Next** part of last node contains **Null** indicating the end of the linked list.



***Q2b. Explain algorithmically the traversal of single linked list.***

***Ans:*** Traversing a linked list refers to visiting each node of the list in order to process the elements stored in the nodes.

**Algorithm:**

Step 1: If Begin = Null Then

Print “Linked List is Empty”

Exit

[End if]

Step 2: Set Pointer = Begin

Step 3: Repeat While Pointer != Null

Print Pointer ---> Info

Assign Pointer = Pointer ---> Next

[End Loop]

Step 4: Exit

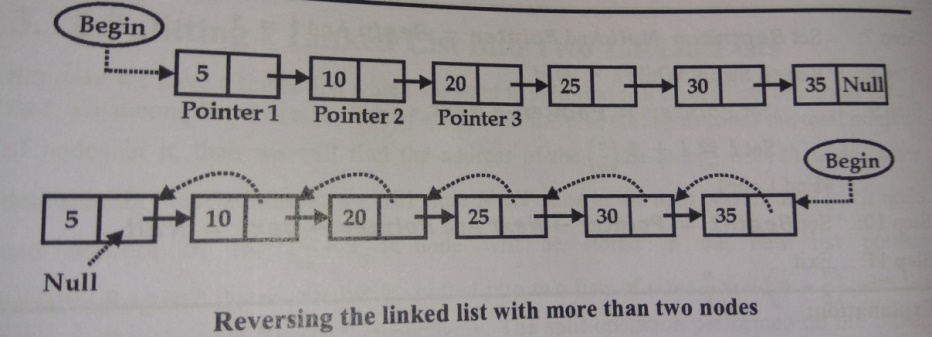
**Explanation**:

In the above algorithm *step* 1 is used to check if the given list is empty. If it is empty print a message and exit. In step 2 variable **Pointer** is assigned address of the first node of the list. Within the loop **Info** part of the currently traversed node is processed and the variable **Pointer** is assigned the address of the next node. These steps will be processed until we come to the **Null** entry in the **Next** part of the node

Mr. Shajil Kumar P A Page 5

***Q2c. Write an algorithm for reversing the single linked list.***

***Ans:*** To reverse a linked list we need to use three pointer variables One pointer variable is used to store the address of current node, second pointer variable is used to store the address of the next node and the third pointer variable is used to store the address of next to next of current node



**Algorithm:**

Step 1: If Begin = Null

Print : “No Node is present in the linked list ”

Exit

[End If]

Step 2: If Begin --> Next = Null Then

Print : “ Linked List is having only one node”

Exit

[End If]

Step 3: If Begin --> Next != Null Then

Set Pointer1 = Begin And Set Pointer2 = Begin --> Next

Set Pointer3 = Pointer2 --> Next

[End If]

Step 4: If Pointer3 = Null Then

Set Pointer2 --> Next = Pointer1 And Set Pointer1 --> Next = Null And Set Begin = Pointer2

Exit

[End If]

Step 5: Set Pointer1 --> Next = Null

Step 6: Repeat Step 7 to 10 while Pointer3 --> Next != Null

Step 7: Set Pointer2 --> Next = Pointer1 And Set Pointer1 = Pointer2

Step 8: Set Pointer2 = Pointer3

Step 9: Set Pointer3 = Pointer3 --> Next

[End Loop]

Step 10: Set Pointer2 --> Next = Pointer1

Step 11: Set Pointer3 --> Next = Pointer2

Step 12: Set Begin = Pointer3

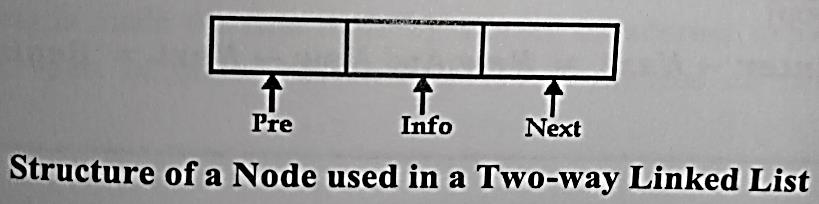
Step 13: Exit

Mr. Shajil Kumar P A Page 6

***Q2d. Explain the structure of double linked list.***

***Ans:*** In certain applications it is required to traverse the list in both directions i.e. in forward direction (from beginning to end) and in backward direction (from end to beginning) This can be accomplished with the help of two-way linked list or doubly linked list. In two-way linked list each node is divided into three parts **Pre, Info, Next** where

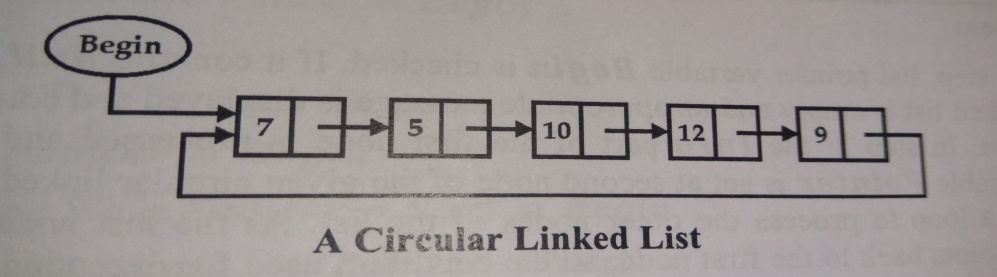
* + **Prev** contains the address of the preceding node
  + **Info** contains the element
  + **Next** contains the address of the next node



In 2-way linked list two list pointers are used **Begin** and **End** which contains the address of the first and the last node respectively. The **Pre** part of the first node of a 2-way linked list will contain **Null** as there is no node preceding the first node and the **Next** part of the last node of a 2-way linked list will contain **Null** as there is no node following the last node

***Q2e. Explain in brief the working mechanism of circular linked list***

***Ans:*** A circular linked list is a list in which last node points back to the first node instead of containing the Null pointer in the next part of the last node. The circular linked list can be shown diagrammatically as



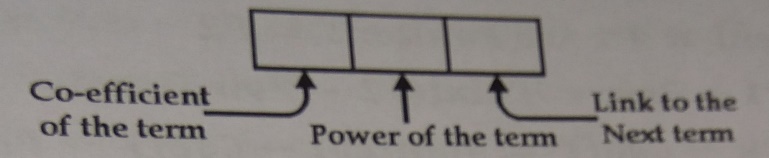
All the operations which can be performed on ordinary singular linked list can easily be performed on circular linked list with the following changes

* In case of 1-way singular linked list the next part of the last node will contain Null address but in case of circular linked list the next part of the last node consist of address of the first node i.e. Begin. Thus for reaching at the end of the circular linked list we will compare the address of the first node i.e. Begin with the address stored in Next part of each node. If both the addresses come out to be same then we have reached at the end of the circular list
* When a new node is to be inserted at the end of the circular linked list its Next part will contain the address of the first node instead of Null as in the case of singular linked list

***Q2f. Explain how polynomials are presented using linked list.***

***Ans***: Polynomials are frequently used in mathematical as well as scientific applications. General purpose languages do not have any built in data structure for storage and manipulation of polynomials and hence linked list can be used to represent the polynomials. Each node of the linked list will have 3 parts where first part of the node will contain the co-efficient of the variable and the second part of the node will contain the power and the third part will contain the address of the next node of the linked list. The structure of the node will be as shown below

Mr. Shajil Kumar P A Page 7



Consider two polynomials P1 and P2 in which we want to subtract P2 from P1

P1: 3x4 – 8x3 + 6x + 9

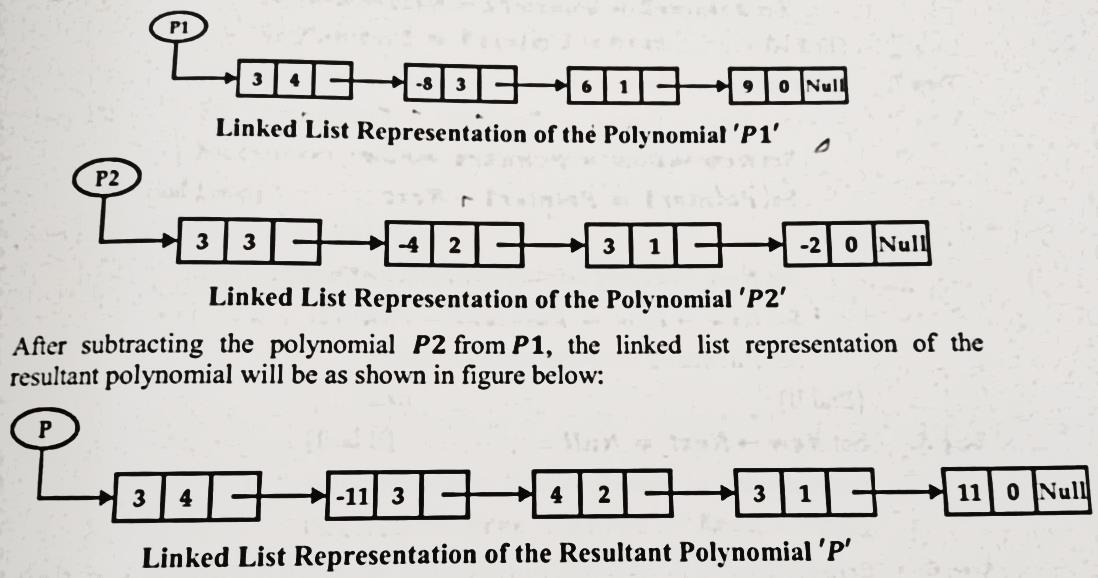
P2: 3x3 – 4x2 + 3x – 2

Their subtraction takes place as follows:

P1: 3x4 – 8x3 + 6x + 9

P2: 3x3 – 4x2 + 3x – 2

Subtraction (P1 – P2) is 3x4 – 11x3 + 4x2 + 3x + 11

The linked list representation of polynomials P1, P2 and (P1-P2) are shown below

***Question 3***

***Q3a. What is stack? Write an algorithm for PUSH operation.***

***Ans:*** Stack is one of the most commonly used linear data structures of variable size. In arrays the insertion and deletion of an element can take place at any position of the array but in the case of stack the insertion and deletion of an element can occur only at one end which is known as TOP In stack insertion operation is known as PUSH and deletion operation is known as POP. Stack is also called Last In First Out (LIFO) list which means that last item added to the stack will be the first item to be removed from the stack. Two basic operations performed on the stack are **Push** and **Pop**

* **Push** operation refers to the insertion of the new element into the stack which will be inserted on the top of the stack
* **Pop** operation refers to the removal of an element from the top of the stack.

Mr. Shajil Kumar P A Page 8

**Algorithm:**

Step 1: If Top = MAX Then

Print “Stack is full Overflow condition”

Exit

[End If]

Step 2: Set Top = Top + 1

Step 3: Set S[Top] = Data

Step 4: Exit

***Q3b. Write the steps for converting infix to postfix And Convert the following expression into a postfix form: a \* b + c + d / (e + f) .***

***Ans:*** **Algorithm:**

Step 1: Push a left parenthesis ( onto the stack

Step 2: Append a right parenthesis ) at the end of the given expression **I**

Step 3: Repeat steps from 4 to 8 by scanning **I** character by character from left to right until the stack is empty

Step 4: If the current character in **I** is a white space, ignore it

Step 5: If the current character in **I** is an operand, write it as the next element of the postfix expression **P**

Step 6: If the current character in **I** is a left parenthesis ( push it onto the stack

Step 7: If the current character in **I** is an operator Then

Pop operators (if there are any) at the top of the stack while they have **equal or higher precedence** than the current operator and put the popped operators in the postfix expression **P**

Push the currently scanned operator on the stack

Step 8: If the current character in **I** is a right parenthesis Then

Pop operators from the top of the stack and insert them in the postfix expression **P** until a left parenthesis is encountered at the top of the stack

Pop and discard the left parenthesis ( from the stack

[End Loop]

Step 9: Exit

**Iin =a \* b + c + d / (e + f)**

= **a \* b + c + d / e f +**

= **a b \* + c + d / e f +**

= **a b \* + c + d e f + /**

= **a b \* c + + d e f + /**

**Ipre** = **a b \* c + d e f + / +**

***Q3c. Explain the working mechanism of Circular queue.***

***Ans:*** Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called as “Ring Buffer”. Operations on Circular Queue are as follows:

**Front:** Get the front item from queue.

**Rear:** Get the last item from queue.

Mr. Shajil Kumar P A Page9

**EnQueue(value)** This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at Rear position.

**Steps:**

* 1. Check whether queue is Full – Check ((rear == SIZE-1 && front == 0) || (rear == front-1)).
  2. If it is full then display Queue is full. If queue is not full then, check if (rear == SIZE – 1 && front != 0) if it is true then set rear=0 and insert element.

**DeQueue()** This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from front position.

**Steps:**

* 1. Check whether queue is Empty means check (front==-1).
  2. If it is empty then display Queue is empty. If queue is not empty then step 3
  3. Check if (front==rear) if it is true then set front=rear= -1 else check if (front==size-1), if it is true then set front=0 and return the element.

***Q3d. Write an algorithm for Deque.***

***Ans:***Double ended queue is the linear queue data structure in which insertion and deletion operations are not restricted to one end but rather insertion and deletion operations can be performed on either of the two ends. These operations cannot be performed at any other position except the two ends of the list. Deque data structure is also known as Deck.

**Algorithm:**

Step 1: If rear=MAX Then

Print "Queue is Overflow”

Else

Rear=Rear+1

q[Rear]=no

If Rear=0

Rear=1;

If Front=0

Front=1;

[End If]

Step 2: If Front<=1Then

Print “Cannot add item at the front”

Else

Front=Front-1;

q[Front]=no;

[End If]

Mr. Shajil Kumar P A Page 10

Step 3: If Front=0

Print "Queue is Underflow”

Else

no=q[Front];

Print “Deleted element is”

If Front=Rear

Front=0

Rear=0

Else

Front=Front+1

[End If]

[End If]

Step 4: If Rear=0

Print “Cannot delete value at rear end”

Else

no=q[Rear];

If Front= Rear

Front=0;

Rear=0;

Else

Rear=Rear-1;

Print “Deleted element”

[End If]

[End If]

Step 5: Exit

***Q3e. Explain the concept of recursion with suitable example.***

***Ans:*** Recursion is a very important and powerful tool for developing algorithms for various problems. Recursion is the ability of the procedure to call itself or calling to some other procedure which may result in call to original procedure. Recursion is generally used for repetitive computation in which each section is defined in terms of previous result. Two important conditions that must be satisfied by a recursive procedure are

* There must be a decision criterion that stops the further call to the procedure called **base criteria**
* Each time a procedure calls itself either directly or indirectly it must be **nearer to the solution** i.e. nearer to the **base criteria**

A procedure having these two properties is called a well-defined procedure and can be defined recursively. Recursive procedure can be implemented in various programming languages like C, C++ etc. Example of a recursion is Factorial function.

The factorial of a positive number n is a product of positive integers from 1 to n

The factorial of a number is represented symbolically by placing a ! next to it

The factorial of a positive number n will be defined as

N! = 1 \* 2 \* 3 \* 4 \* ………….. \* (n-1) \* n

The formal definition of factorial function can be given as

n! =

Mr. Shajil Kumar P A Page 11

**Algorithm**: Calculate the value of n! recursively

Factorial(n)

If n = 0 Then

Set Fact = 1

Return

Else

Set Fact = n \* Factorial (n – 1)

Return

[End If]

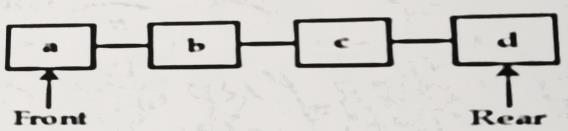
***Q3f What is Queue? Explain the operations of queue with suitable example.***

***Ans:*** Queue is a linear collection of elements in which insertion takes place at one end known as rear and deletion takes place at another end known as front of the queue. Queue is also a restricted data structure like stack because the new element can be added at its one end known as rear of queue and element can be removed from the other end known as front of the queue. Queue is known as **First In First Out (FIFO)** list. The element of a general queue is processed on **First Come First Serve basis (FCFS)**. Two basic operations performed on the queue are **Insertion** and **Deletion**

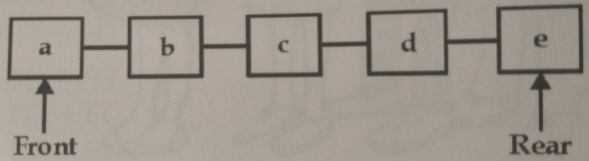
**Insertion:** The insertion operation refers to the addition of a new element at the rear end of the queue. Insertion operation can be performed only when queue has space to accommodate the new element. The condition of attempting to insert an element in a queue having no space results in a state called overflow condition.

**Deletion:** Deletion refers to the removal of an element from the front of the queue.

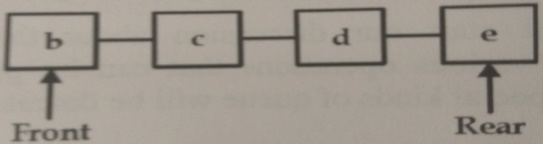
Deletion operation can be performed only when the queue is not empty. The condition of attempting to delete an element from an empty queue is known as underflow condition. Consider a list of four elements **(a, b, c, d)** where **a** is the front element and **d** is the rear element. The queue of these elements can be represented diagrammatically as shown below



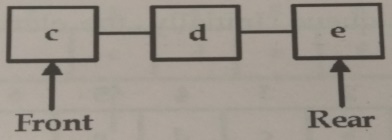
New element e will be inserted at the rear end in the given queue after element d as shown below



Now element **a** will be deleted from the front end of the queue



Mr. Shajil Kumar P A Page 12

Now let us delete another element from the queue. It will be b as b is the front element in the queue

***Question 4***

***Q4a Write an algorithm for Bubble Sort.***

***Ans:*** Sorting an array refers to arranging the elements of an array in some logical order, which can be increasing or decreasing of elements. The sorting technique which is popularly used is **Exchange Sort or Bubble Sort**. The algorithm starts with comparing the first element with the second element and exchange takes place only if the first element is larger than the second element. Then the second element is compared with the third element and so on. This process goes on until the array is sorted

**Algorithm**

Step 1.Repeat For p=1 to n-1

Step 2. For i=1 to n-p

Step 3. If S[i]>S[i+1] Then

Exchange S[i] with S[i+1]

[End If]

[End Loop]

[End Loop]

Step 4. Exit

***Q4b Explain the difference between binary search and sequential search.***

***Ans:***

|  |  |
| --- | --- |
| **Linear Search** | **Binary Search** |
| The elements of the array need not to be in sorted order. | The elements of the array must be in any sorted order. |
| The complexity of linear search is O(n) | The complexity of binary search is O(log2n) |
| Linear search is accomplished by comparing desired element with each element of the list(array) starting from 1st element of the array | In binary search, the desired element is compared with middle element and selecting the half portion of the list in which element may be present. This procedure of halving the list is repeated till the element is found or we conclude that element is not present. |
| Linear search can be applied on any linear data structure even if the elements of the data structure do not occupy the contiguous memory locations. | The binary search can be applied only on array because the elements of array are in contiguous memory locations. |

Mr. Shajil Kumar P A Page 13

***Q4c What is heap? Explain the concept of minimum heap***

***Ans:*** Heap is a very important data structure which can be used efficiently to sort given list of elements which is maintained in a memory using a linear array. A heap or **max heap** is binary tree which satisfies the following characteristics

* The binary tree should be almost complete i.e. all the leaf nodes should be at **kth or (k+1)th** level
* The value at any node is larger than or equal to the value at each of its two children. This means the element in parent node is always larger than or equal to the value at each of its two children. This means the elements in the parent node is always larger than or equal to the elements in child nodes

There also exists a **min heap** in which the element at each node is less than or equal to the elements at its child nodes. Consider the binary tree shown below which is a min heap



***Q4d Sort the following elements using Insertion sort. 22, 43, 12, 55, 67, 71, 5, 89, 47, 50***

***Ans:*** Consider the given array as shown below

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **22** | 43 | 12 | 55 | 67 | 71 | 5 | 89 | 47 | 50 |

1st Pass

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **22** | **43** | 12 | 55 | 67 | 71 | 5 | 89 | 47 | 50 |

2nd Pass

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **12** | **22** | **43** | 55 | 67 | 71 | 5 | 89 | 47 | 50 |

3rd Pass

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **12** | **22** | **43** | **55** | 67 | 71 | 5 | 89 | 47 | 50 |

4th Pass

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **12** | **22** | **43** | **55** | **67** | 71 | 5 | 89 | 47 | 50 |

5th Pass

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **12** | **22** | **43** | **55** | **67** | **71** | 5 | 89 | 47 | 50 |

6th Pass

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **5** | **12** | **22** | **43** | **55** | **67** | **71** | 89 | 47 | 50 |

7th Pass

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **5** | **12** | **22** | **43** | **55** | **67** | **71** | **89** | 47 | 50 |

8th Pass

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **5** | **12** | **22** | **43** | **47** | **55** | **67** | **71** | **89** | 50 |

9th Pass

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **5** | **12** | **22** | **43** | **47** | **50** | **55** | **67** | **71** | **89** |

Mr. Shajil Kumar P A Page 14

***Q4e What is binary tree? Construct the binary tree for the following: 21, 18, 7, 9, 11, 8, 19, 14, 13, 6.***

***Ans***: A binary tree can be defined as a finite collection of nodes where each node is having the property that it can have **0, 1, or 2 children**. A binary tree may be empty known as **null tree**. A binary tree contains a special node called the root of the tree and remaining nodes in the tree form the left and right binary subtree.

**Add 21 Add 18 Add 7 Add 9**

**Add 11** **Add 8**

**Add 19**

Mr. Shajil Kumar P A Page 15

**Add 14**

**Add 13**

**Add 6**

Mr. Shajil Kumar P A Page 16

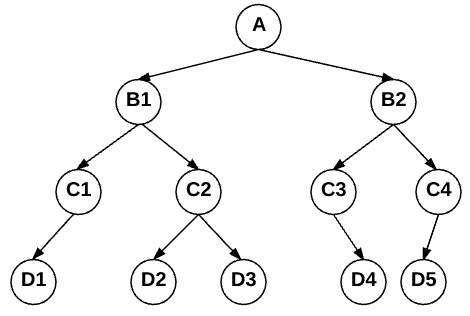
***Q4f Explain In-Order and Pre-Order traversal of the tree.***

***Ans:* In-Order** - In in-order traversal (also known as **symmetric order**), firstly, whole left subtree is traversed recursively in the in-order traversal. This traversal method is also known as **Left-Root-Right** traversal. In this method, traversal order is as follows:

* Traverse the left subtree in in-order traversal
* Visit the root
* Traverse the right subtree in in-order traversal

**Pre-Order** - In pre-order traversal (also known as **depth – first order**), first we visit the root then recursively perform the pre-order traversal on the left sub-tree and then perform the pre-order traversal on the right subtree. This traversal method is also known as **Root-Left-Right** traversal. In this method, traversal order followed is:

* Visit the root
* Traverse the left subtree in pre-order traversal
* Traverse the right subtree in pre-order traversal



The **pre-order** traversal of the tree shown in figure will result in the following processing order:

**a b1 c1 d1 c2 d2 d3 b2 c3 d4 c4 d5**

The **in-order** traversal of the tree shown in figure will result in the following processing order:

**d1 c1 b1 d2 c2 d3 a c3 d4 b2 d5 c4**

***Question 5***

***Q5a What is Hashing? Explain Linear Probing with suitable example***

***Ans:*** The idea of calculating the physical address of the records by using their keys can be implemented by using a variety of methods known as hashing techniques. Hashing is mainly a searching technique. However the main aim of all the hashing functions is to map a relatively large domain of key values to a relatively small range of addresses.

**Linear Probing:** If a record with key **K** is mapped to an address which is already occupied by some other record then linear search starting from address generated by hash function is done until a free location is found for the storage of the new record. An empty location is always met if it is available. Linear probing algorithms are implemented in such a way that, starting from the address generated by the hash function, the status of each location is checked whether it is empty or not. Once an empty location is encountered, record is stored over there. On the other hand if the end of the address space is reached without meeting any free location then rather quitting, we starts looking for the empty location from the first address of the address space until a free location is encountered or location having address which was generated by the hash function is encountered again.

Mr. Shajil Kumar P A Page 17

For using linear probing technique the hash function will be:

**Hash(K , p) =[H(K)+p]mod m**

Here **H(K)** is the Hash function used to find the relative address before collision.

So, **H(K)= K mod m**

**p** is the probe number which can be **0,1,2,…. m-1, and m** is the size of the hash table. Consider the table of size 10. Suppose we want to insert some record with key values 33,101,99,83,93

Here, The hash function is taken as

**H(K)= K mod 10**

**For 1st Key, k = 33**

Hash (K, p) = [H(K)+p]mod m

Hash (33, 0) = (33 mod 10 +0) mod 10 = 3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 33 |  |  |  |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

**For 2nd Key, k = 101**

Hash(K , p)= [H(K)+p]mod m

Hash(101,0)= (101 mod 10 +0)mod 10 = 1 mod 10 =1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 101 |  | 33 |  |  |  |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

**For 3rd Key, k = 99**

Hash(K , p)= [H(K)+p]mod m

Hash(99,0)= (99 mod 10 +0)mod 10 = 9 mod 10 =9

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 101 |  | 33 |  |  |  |  |  | 99 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

**For 4th Key, k = 83**

Hash(K , p)= [H(K)+p]mod m

Hash(83,0)= (83 mod 10 +0)mod 10 = 3 mod 10 = 3

The record with this key (k=83) should be at index 3 of hash table but this position is already occupied so we increase the probe by 1. (i.e p=1) and again find the address

Hash(K, p)=[H(K)+p]mod m

Hash(83,1)= [83 mod 10 +1]mod 10 = (3+1) mod 10 = 4

So, this record with key (K=83) will be stored at 4th position of the table.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 101 |  | 33 | 83 |  |  |  |  | 99 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

**For 5th Key, k = 93**

Hash (K, p) = [H (K)+p]mod m

Hash (93, 0) = (93 mod 10 +0)mod 10 = 3 mod 10 = 3

But the 3rd position in the hash table is already occupied so we increase the probe by 1. (i . e p=1)

Hash (K, p)=[H(K)+p]mod m

Hash (93,1)= [93 mod 10 +1]mod 10 = (3+1) mod 10 = 4

Again, 4th position in the hash table is already occupied so we increase the probe by 1.

Hash (K, p)=[H(K)+p]mod m

Hash (93, 2)= [93 mod 10 +2]mod 10 (3+2) mod 10 = 5

So, this record with key(K=93) will be stored at 5th position of the hash table.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 101 |  | 33 | 83 | 93 |  |  |  | 99 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Mr. Shajil Kumar P A Page 18

***Q5bWhat is collision? Explain how it is resolved.***

***Ans:*** Hashing function maps a large key space to a relatively small address space, so there are possibilities that more than one different key value can be mapped to the same address.

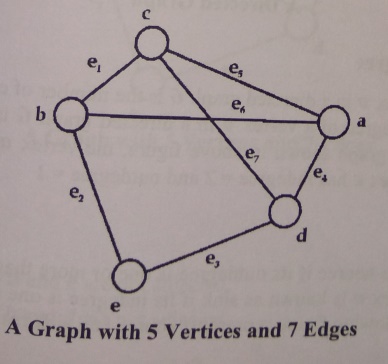
Consider the situation when we want to store a new record with key value K and the address generated by the hash function H(k), is already occupied by some other record with the key value Ki. This situation is called collision and the address for the colliding record will now be decided by some collision resolution technique. In this subsection, the two categories of collision resolution techniques are:

**Open Addressing**: In this category of resolving the collision of the records, the colliding record’s key is mapped to some other vacant address in the relative file. If no such free address in found then the overflow condition occurs and the key of record cannot be mapped to relative address. The different techniques to find the free slot is called probing or open addressing. Main techniques of open addressing are: Linear Probing, Quadratic Probing, Double Hashing.

**Chaining**: One of the most efficient methods of handling colliding records is called synonym chaining. In this approach, colliding records are chained together by maintaining a linked list of such colliding records. A separate one-way list is maintained for each set of records which are colliding to the same location. The records in these lists are not kept according to any specific order but whenever a new collision occurs, the colliding record is inserted at the front of the appropriate linked list. In this approach of handling collision, instead of storing records at locations in the address space, pointers are placed where each pointer points to the chain of records which shares the same hash location.

***Q5c What is Graph? Explain directed and undirected graph.***

***Ans:*** Graph is a non-linear data structure which finds its application in various engineering domains. A graph consists of finite set of vertices **VG** and finite set of edges **EG** which can be denoted by a tuple **G = (VG,EG)**. Here set of vertices **VG** represent the entities which has names and some other attributes. An edge connects a pair of vertices and represents a relationship between the two entities. A graph may be pictorially represented as shown in the following figure

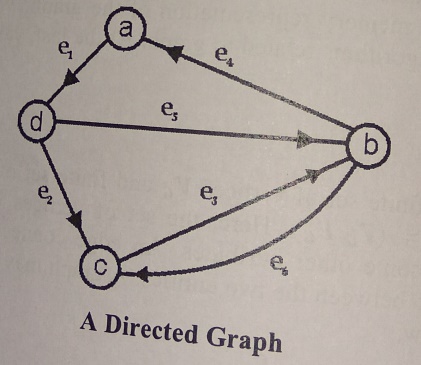


In this graph, vertices are labeled using letters a, b, c, d and e. Therefore

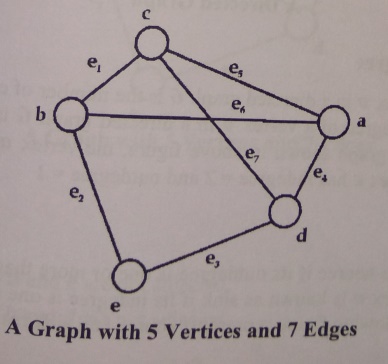
* + VG = {a, b, c, d, e}
  + EG = {ab, be, ed, dc, ca, bc, ad}
  + EG = {e1, e2, e3, e4, e5, e6, e7}

**Directed Graph** - In directed graph as shown in figure below each edge is assigned a direction or we can say that each edge is identified by an ordered pair of vertices .The edge is represented as E{ad} where the edge starts at vertex a and ends at vertex d

Mr. Shajil Kumar P A Page 19



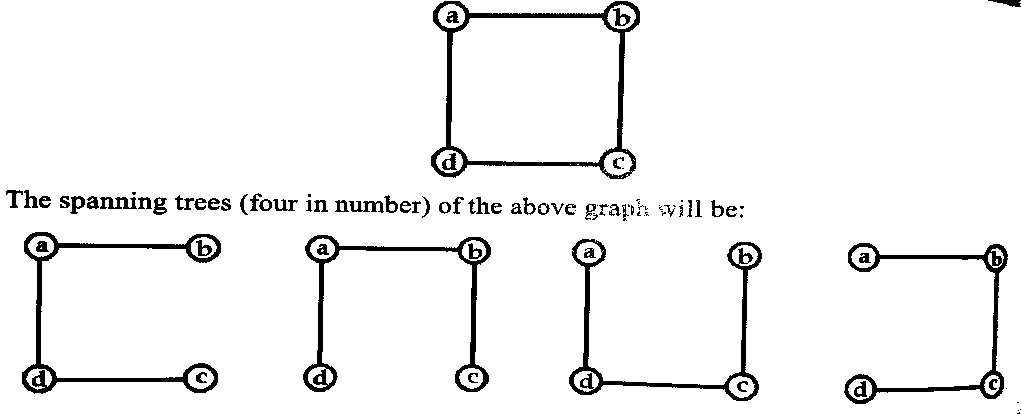
**Undirected Graph** - When there are no directions are associated with the edges of the graph. So this graph is an undirected graph. The edges of an undirected graph are represented by unordered pair of vertices. The edge is represented as E{ad} or E{da} where the edge starts or ends at vertex a and starts or ends at vertex d

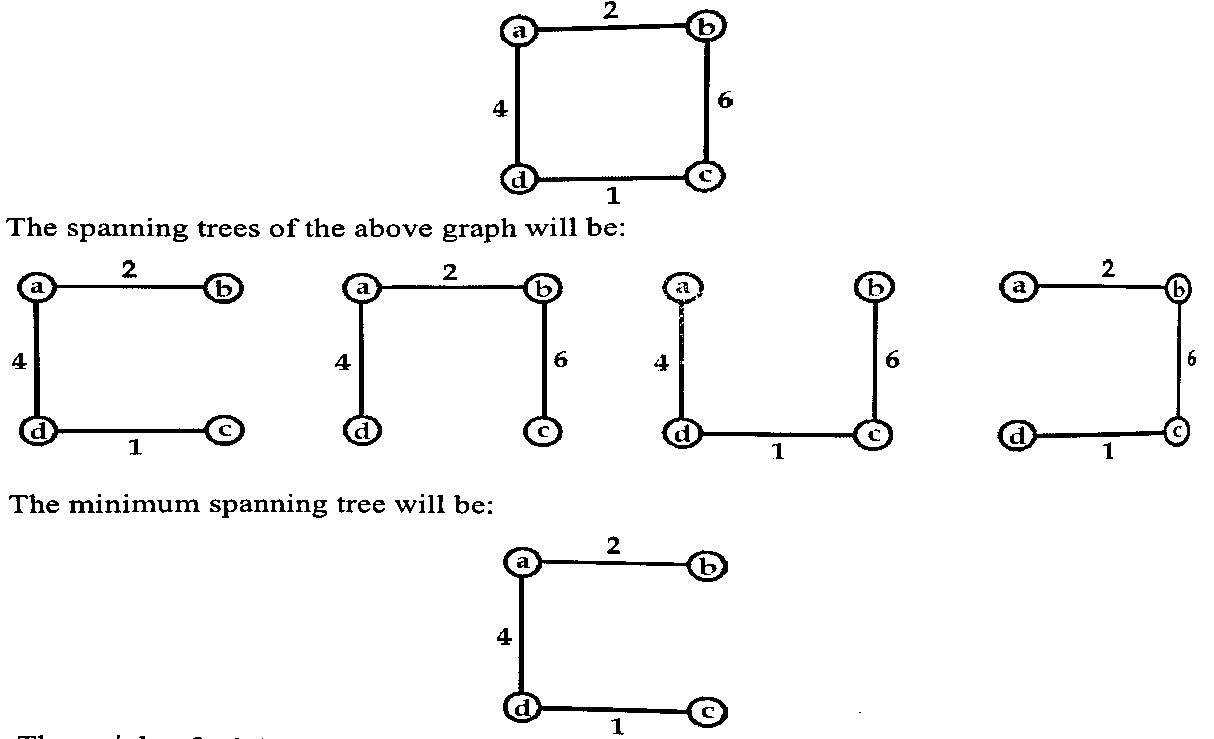


***Q5d Explain in brief about spanning tree with suitable example***

***Ans:*** A tree can be defined as **undirected, acyclic** and **connected graph** if there is only one path connecting each pair of vertices. A spanning tree of a connected graph **G** is a tree that contains all the vertices of graph **G**. We can also define a spanning tree of a connected graph **G** as a **subgraph** of **G** that contains all the vertices and is a tree.

A connected graph **G** may have spanning trees. For example, consider a graph **G** as shown below:



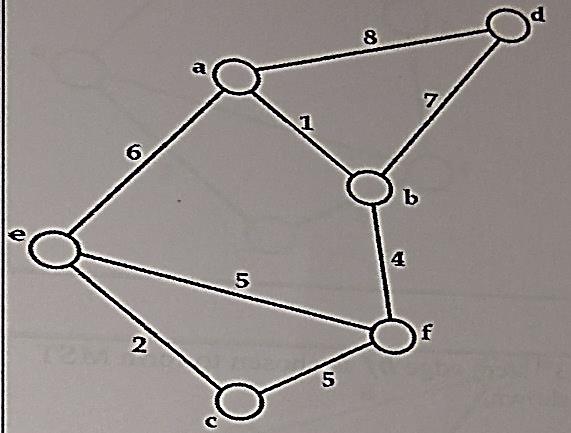
******If we consider a weighted graph then a minimum spanning tree may be defined as a spanning tree with minimum weight. Here the weight of the tree is sum of weight of its edges. Consider graph as shown below. The spanning graph and its spanning trees are shown below

Mr. Shajil Kumar P A Page 20

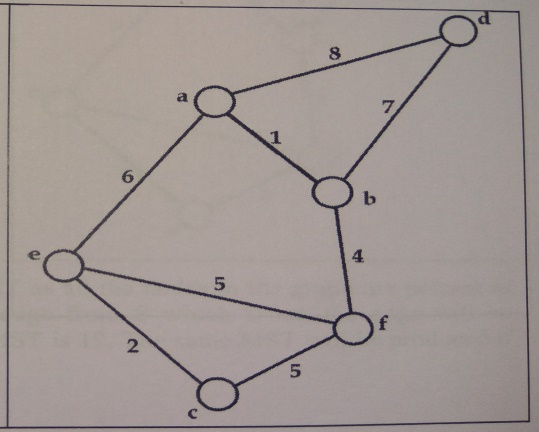
***Q5e Give the outline of Kruskal’s algorithm.***

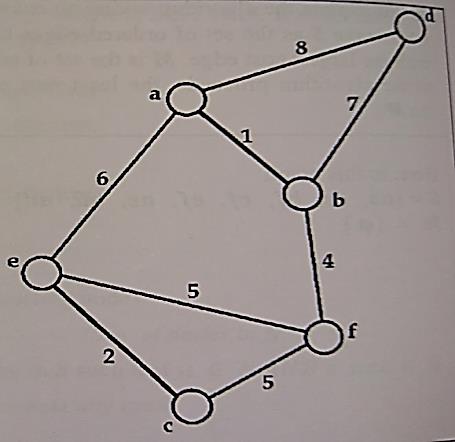
**Ans:** In Kruskal’s algorithm all the edges of the graph **G** are ordered in increasing order of their weight. Then at each step, we add the least cost edge to make MST so that it joins the two partial trees together. If adding an edge forms a cycle, it is rejected as in this case; both the node of the edges is already in the partial MST formed. Consider a graph **G**. Here, we assume **S** as the set of ordered edges beginning with the least cost edge and ending with the largest cost edge. **M** is the set of edges giving the MST. Initially **M** is empty and as the algorithm proceeds, the least cost edge is added to the **M**. Here in this graph **S= {ab, ce, bf, cf, ef, ae, bd, ad}**

**M= {ɸ}**

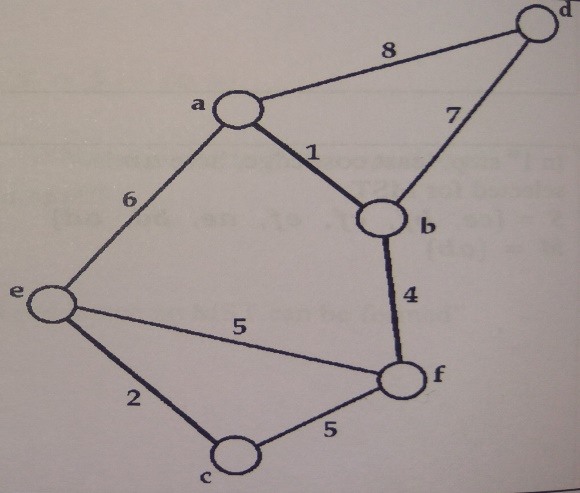
****

In 1st step least cost edge here ab is selected for MST. **S= {ce,bf,cf,ef,ae,bd,ad} M= {ab}**

****

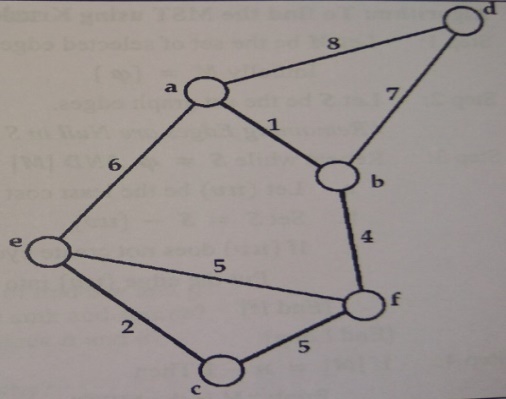
In 2nd step least cost edge here ce is selected for MST **S= {bf,cf,ef,ae,bd,ad} M= {ab, ce}**

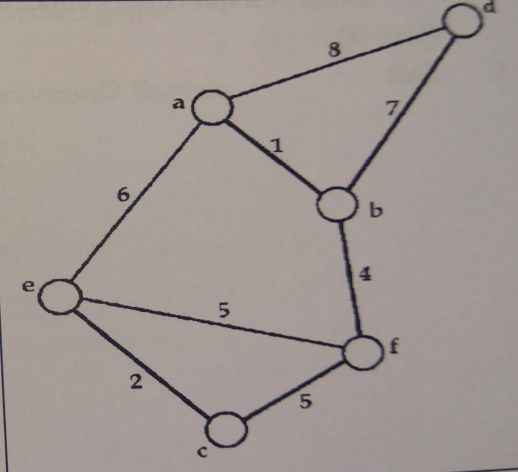
In 3rd step least cost edge here **bf** is selected for MST **S= {cf,ef,ae,bd,ad} M= {ab, ce, bf}**



Mr. Shajil Kumar P A Page 21

In 4th step least cost edge here cf is selected for MST **S= {ef,ae,bd,ad} M= {ab, ce, bf, cf}**

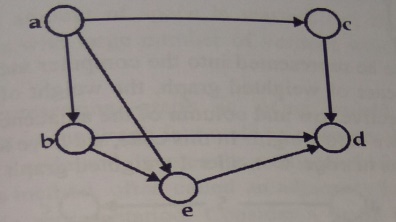


In 5th step least cost edge here **bd** is selected for MST because the other edges ef and ae are rejected because they create cycle **S= {ad} M= {ab, ce, bf, cf, bd}**

***Q5f What is Adjacency Matrix? Generate adjacency matrix for the following undirected graph***

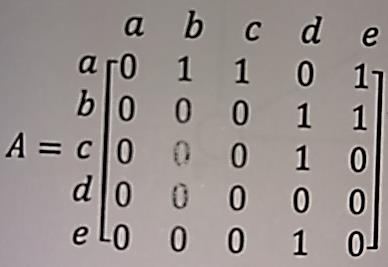
***Ans:*** Suppose G = (Vg, Eg) is a directed graph having n nodes. Suppose vertices are ordered by using v1, v2, v3, v4,…….vn, then the adjacency matrix A for the graph G will be a square matrix of order n such that

aij =

The adjacency matrix of a graph depends upon the ordering of its vertices that is if we change the order of vertices then it will result in different adjacency matrix. Consider the graph given below

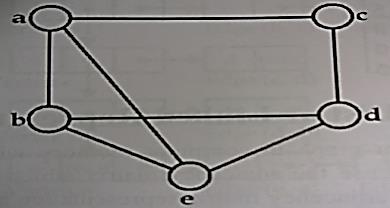
**VG = {a, b, c, d, e}**

The adjacency matrix corresponding to this ordering sequence will be



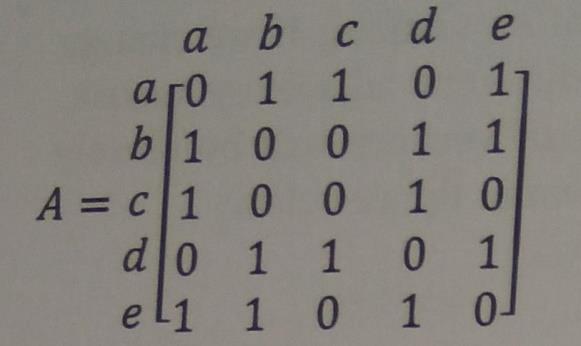
Mr. Shajil Kumar P A Page 22

In case of undirected graph the adjacency matrix will be symmetric as there will be two entries in the matrix corresponding to each edge in the graph. Consider the graph given below



**VG = {a, b, c, d, e}**

The adjacency matrix corresponding to this ordering sequence will be



Mr. Shajil Kumar P A Page 23